UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ELCIO FAGUNDES BOIN

GERAÇÃO DE MAPAS 2D COM LLMS E VALIDAÇÃO BASEADA EM AGENTES

CHAPECÓ
2026

ELCIO FAGUNDES BOIN

GERAÇÃO DE MAPAS 2D COM LLMS E VALIDAÇÃO BASEADA EM AGENTES

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal da Fronteira Sul (UFFS), como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Giancarlo Dondoni Salton

CHAPECÓ
2026

Bibliotecas da Universidade Federal da Fronteira Sul - UFFS

Boin, Elcio Fagundes
Geração de Mapas 2D com LLMs e Validação Baseada em Agentes / Elcio Fagundes Boin. -- 2026.
18 f.:il.

Orientador: Giancarlo Dondoni Salton

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal da Fronteira Sul, Curso de Bacharelado em Ciência da Computação, Chapecó,SC, 2026.

1. Geração Procedural de Conteúdo. 2. Modelos Grandes de Linguagem. 3. Geração de Mapas 2D. 4. Validação Baseada em Agentes. I. Salton, Giancarlo Dondoni, orient. II. Universidade Federal da Fronteira Sul. III. Título.

Elaborada pelo sistema de Geração Automática de Ficha de Identificação da Obra pela UFFS com os dados fornecidos pelo(a) autor(a).

“//pra que serve
//tantos códigos?
//se a vida
//não é programada
//e as melhores coisas
//não tem lógica”
(Desconhecido)

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação, 2026
Universidade Federal da Fronteira Sul, Campus Chapecó

Revisão da Literatura

Geração de Mapas 2D com LLMs e Validação Baseada em Agentes

2D Map Generation with LLMs and Agent-Based Validation

Elcio Fagundes Boin [Universidade Federal da Fronteira Sul | *elcio_fagundes@estudante.uffs.edu.br*]
Giancarlo D. Salton [Universidade Federal da Fronteira Sul | *gian@uffs.edu.br*]

Universidade Federal da Fronteira Sul, Campus Chapecó, Rodovia SC 484, km 02, Bairro Fronteira Sul, Chapecó, Santa Catarina, 89815-899, Brasil.

Resumo. Este trabalho investiga o uso de LLMs na geração procedural de mapas 2D para jogos *dungeon crawler*, combinando criatividade guiada por *prompts* com validação determinística baseada em agentes. Propõe-se um pipeline híbrido: (i) geração de uma matriz ASCII em JSON com regras de dimensões, legenda e restrições topológicas; (ii) normalização da grade e *auto-tiling* de paredes; e (iii) triagem por um agente validador com verificações sintáticas e testes de conectividade/acessibilidade via A^*. Um protótipo na Godot 4.6 foi avaliado em campanhas variando tamanho do mapa e densidade de entidades, comparando Gemma 4, Ministral 3 e Gemini 3.1 Flash Lite. Os resultados indicam elevada instabilidade na saída bruta, com falhas críticas associadas à "cegueira espacial" e à contagem imprecisa; porém, o ciclo de retentativas com validação recupera mapas jogáveis e mitiga *soft-locks*. A avaliação Likert aponta boa coerência temática (especialmente em modelos locais), enquanto a latência limita uso em tempo real, favorecendo pré-produção.

Abstract. This paper investigates LLMs for procedural generation of 2D *dungeon crawler* maps, combining prompt-guided creativity with deterministic agent-based validation. We propose a hybrid pipeline: (i) strict JSON-based ASCII grid generation with explicit dimension/entity/topology rules; (ii) grid normalization and wall *auto-tiling*; and (iii) screening via an A^*-based validator for syntactic, connectivity, and accessibility constraints. A Godot 4.6 prototype was evaluated across campaigns with varying map sizes and entity densities, comparing Gemma 4, Ministral 3, and Gemini 3.1 Flash Lite. Raw LLM outputs proved unstable, with critical failures driven by spatial blindness and poor counting; however, the retry-and-validate loop recovers playable layouts and mitigates *soft-locks*. A Likert study suggests good thematic coherence (notably for local models), while inference latency limits real-time use, favoring offline pre-production.

Palavras-chave: Geração Procedural de Conteúdo; Modelos Grandes de Linguagem; Geração de Mapas 2D; Validação Baseada em Agentes.

Keywords: Procedural Content Generation; Large Language Models; 2D Map Generation; Agent-Based Validation.

1 Introdução

O desenvolvimento de jogos digitais tem sido impulsionado pela demanda por experiências interativas, imersivas e personalizadas, o que tem ampliado o uso de técnicas de Inteligência Artificial (IA) no processo de criação [Hendrikx *et al.*, 2013]. Nesse contexto, a Geração Procedural de Conteúdo (GPC) se destaca por reduzir custos e aumentar a rejogabilidade (*replayability*), automatizando a criação de níveis, mapas, personagens e desafios [Maleki and Zhao, 2024].

Tradicionalmente, a criação de níveis e mapas é uma das tarefas mais complexas e demoradas do desenvolvimento de um jogo [Medeiros, 2023]. Ela depende do trabalho manual de designers, que precisam equilibrar dificuldade, fluidez e coerência estética. A automação parcial via GPC possibilita mundos virtuais variados sem reconstrução manual a cada nova partida, sendo comum em gêneros como *roguelike* e *dungeon crawler* [Viana, 2019].

Apesar de sua utilidade, abordagens tradicionais de GPC têm limitações. A primeira é a dificuldade de garantir, de forma consistente, a jogabilidade do conteúdo gerado: métodos baseados em ruído ou aleatoriedade podem criar falhas estruturais, como áreas inacessíveis ou desafios impossíveis, especialmente em jogos de plataforma e *roguelikes*. A segunda é a falta de coerência semântica e narrativa: algoritmos clássicos produzem estruturas, mas não interpretam tema ou intenção de design, resultando em níveis genéricos.

Com o avanço da IA, surgiram alternativas para mitigar essas restrições. Entre elas, destacam-se os *Large Language Models* (LLMs)[1], capazes de interpretar instruções em linguagem natural e gerar saídas estruturadas e semanticamente coerentes. Segundo Maleki and Zhao [2024], essas arquiteturas, baseadas no modelo *Transformer* [Vaswani *et al.*, 2017], mostram potencial para compreender contextos complexos e produzir resultados criativos, abrindo caminho para sua aplicação na geração de conteúdo procedural em jogos.

Assim, este trabalho propõe investigar e desenvolver um sistema de geração adaptativa de mapas para jogos digitais, no qual LLMs realizam a criação e um agente autônomo realiza a validação. Inspirado por técnicas de *automated playtesting*, esse agente analisará a estrutura do mapa gerado usando algoritmos de busca de caminho, como o A^* (A-Estrela), para certificar-se de que o nível é solucionável e cumpre regras de design. Parte-se da hipótese de que as LLMs podem superar a limitação de coerência temática ao interpretar *prompts* em linguagem natural que descrevem não apenas a estrutura, mas também o tema e a atmosfera do mapa, permitindo uma geração não apenas procedural, mas criativamente direcionada.

[1] Em português, Modelos Grandes de Linguagem.

2 Jogos

Os jogos digitais representam uma das formas de entretenimento mais relevantes e lucrativas da era moderna [Bento, 2018]. Desde seus primórdios em laboratórios de computação na década de 1950, os jogos evoluíram de simples experimentos para produções complexas, integrando elementos técnicos e artísticos para criar experiências interativas e imersivas. Um jogo pode ser definido como um sistema composto por regras, objetivos e participantes, no qual a interação do jogador com seus elementos é fundamental [Canedo, 2022; de Oliveira, 2022].

A evolução do hardware e das ferramentas de desenvolvimento intensificou a competição no mercado e elevou as expectativas por conteúdo de alta qualidade. Nesse cenário, técnicas de Inteligência Artificial (IA) tornaram-se centrais para criar adversários desafiadores e enriquecer a experiência do jogador com mundos dinâmicos e conteúdo personalizado [Barbosa, 2024; Kühr, 2018].

A indústria de jogos digitais é vasta e diversificada, abrangendo múltiplos gêneros que categorizam os jogos com base em suas mecânicas de interação, objetivos e estética. Esses gêneros alinham as expectativas dos jogadores e guiam as decisões dos desenvolvedores. Entre os mais proeminentes, destacam-se FPS, RTS e Jogos de Interpretação de Papéis (RPG). Cada gênero apresenta desafios próprios de design e jogabilidade, além de um campo de estudo para a aplicação de tecnologias como a Inteligência Artificial [Kim *et al*., 2010].

O termo roguelike (em português, "similar a Rogue") descreve um subgênero de RPGs cujas fundações foram estabelecidas pelo jogo Rogue, de 1980. Sua característica mais marcante é o uso intensivo da GPC (do inglês *Procedural Content Generation*) para criar mapas, itens e a disposição de inimigos de forma algorítmica a cada nova partida [de Oliveira, 2022]. Assim, nenhuma sessão é idêntica à anterior, elevando a rejogabilidade ao exigir adaptação constante [Souza *et al*., 2019].

A GPC utiliza algoritmos para criar conteúdo de jogo de forma automática, com pouca ou nenhuma intervenção humana direta [de Oliveira, 2022]. Embora possa ser aplicada a itens, narrativas e personagens, sua aplicação mais difundida é na criação de conteúdo espacial, isto é, na geração de mapas e níveis [Maleki and Zhao, 2024]. Seu objetivo é aliviar a carga de trabalho manual dos designers e aumentar a rejogabilidade, oferecendo ambientes novos e únicos a cada partida [de Pontes and Gomes, 2020].

Diferente de uma geração puramente aleatória, que pode resultar em conteúdo caótico ou injogável, a GPC opera a partir de regras e heurísticas que guiam o processo, buscando um equilíbrio entre variedade e coerência [Medeiros and Rodrigues, 2025]. A literatura acadêmica e a indústria consolidaram diversas técnicas para essa tarefa.

Uma das abordagens mais fundamentais é a geração baseada em Ruído (Noise). Algoritmos como o Ruído de Perlin criam um mapa de valores pseudoaleatórios contínuos, interpretados para gerar terrenos com aparência natural. Essa técnica é amplamente utilizada por ser computacionalmente leve e adequada para ambientes abertos [Canedo, 2022].

Para a criação de mapas de masmorras e cavernas, uma técnica popular são os Autômatos Celulares (*Cellular Automata*). Nesse método, um mapa é representado como uma grade de células, e regras simples de vizinhança são aplicadas iterativamente para determinar se uma célula se torna, por exemplo, uma parede ou um chão. Após algumas iterações, padrões que se assemelham a cavernas emergem, criando layouts não-lineares para exploração [de Oliveira Barbosa Leite and de Lima, 2015; Hendrikx *et al*., 2013].

Quando a estrutura e a conectividade do mapa são mais importantes, como em masmorras com progressão lógica específica, a técnica de Gramática de Grafos se destaca. Nessa abordagem, o mapa é representado como um grafo, onde os nós são as salas e as arestas são as conexões. Regras de produção expandem o grafo, adicionando salas e corredores para garantir uma estrutura coesa e alinhada a uma lógica de design predefinida [Souza *et al*., 2019; Maleki and Zhao, 2024].

Além desses métodos, abordagens híbridas também são utilizadas. Jogos como *Diablo* popularizaram a técnica de gerar mapas combinando salas pré-desenhadas, conectadas proceduralmente por corredores, preservando qualidade local e variedade global [de Oliveira Barbosa Leite and de Lima, 2015].

Embora essas técnicas clássicas sejam poderosas, elas operam a partir de regras e lógicas explicitamente programadas. O surgimento das LLMs representa uma nova fronteira para a GPC: esses modelos podem gerar conteúdo a partir de descrições em linguagem natural, interpretando não apenas regras estruturais, mas também intenções temáticas e narrativas. A proposta deste trabalho se insere nessa vanguarda, investigando como as LLMs podem ser utilizadas para gerar o próprio mapa de forma criativa e contextualizada.

3 LLMs

Nos últimos anos, os LLMs se consolidaram como o principal modelo baseado em redes neurais artificiais em tarefas de Processamento de Linguagem Natural (PLN) e estão presentes em ferramentas comerciais como o ChatGPT e o Gemini. Diferente das IAs tradicionais, geralmente especializadas em tarefas restritas, as LLMs demonstram uma capacidade notável de gerar e manipular texto de forma coerente e contextualizada, aproximando-se da fluidez humana [Rolim, 2023].

Um LLM é um modelo computacional treinado para calcular a probabilidade de uma sequência de palavras; sua tarefa fundamental é prever a próxima palavra (ou *token*) em um dado texto [Vaswani *et al*., 2017]. De acordo com Salton *et al*. [2017], um modelo de linguagem produz a probabilidade conjunta de uma sequência $w_1, ..., w_N$ utilizando a regra da cadeia de probabilidades, definida como:

$$p(w_1, \ldots, w_N) = \prod_{t=1}^{N} p(w_n \mid w_1, \ldots, w_{n-1}) \qquad (1)$$

onde: $w_1, ..., w_N$ é a sequência de palavras; N é o número de palavras; e $p(w_n|w_1, ..., w_{n-1})$ representa a probabilidade da palavra w_n ocorrer, dada a sequência anterior. Segundo Zhao *et al*. [2025], modelos mais simples, como os baseados em *n-grams*, faziam essa estimativa por frequência, enquanto o avanço das redes neurais permitiu capturar relações mais complexas e de longa distância, resultando em uma compreensão

de contexto mais profunda.

A transição para as LLMs foi marcada por três fatores: (i) o aumento massivo no número de parâmetros do modelo; (ii) o aumento no volume de dados de treinamento; e (iii) o aumento no poder computacional utilizado [Zhao *et al*., 2025]. Esse ganho de escala favoreceu o surgimento das chamadas "habilidades emergentes" (*emergent abilities*), como a capacidade de seguir instruções detalhadas e responder de forma mais lógica.

O processo de criação de um LLM começa com o pré-treinamento, no qual o modelo é exposto a uma grande quantidade de dados textuais (conteúdos da internet, livros digitalizados e outros acervos). Nessa fase, aprende regras gramaticais, fatos sobre o mundo e padrões semânticos e contextuais da linguagem, formando uma base generalista que pode ser adaptada a tarefas específicas, como a geração de mapas proposta neste trabalho.

Antes de 2017, o estado da arte para o processamento de sequências era dominado por arquiteturas recorrentes, como RNNs (*Recurrent Neural Network*) e *Long Short-Term Memory* (LSTMs). Esses modelos processam os dados de forma sequencial e mantêm um "estado de memória" para carregar o contexto, mas sofriam com a dificuldade de capturar dependências de longa distância e com a baixa paralelização, tornando o treinamento de modelos muito grandes lento e computacionalmente caro [Rolim, 2023].

A arquitetura *Transformer* propôs um LLM que abandonou a recorrência [Vaswani *et al*., 2017]. A Figura 1 ilustra sua estrutura.

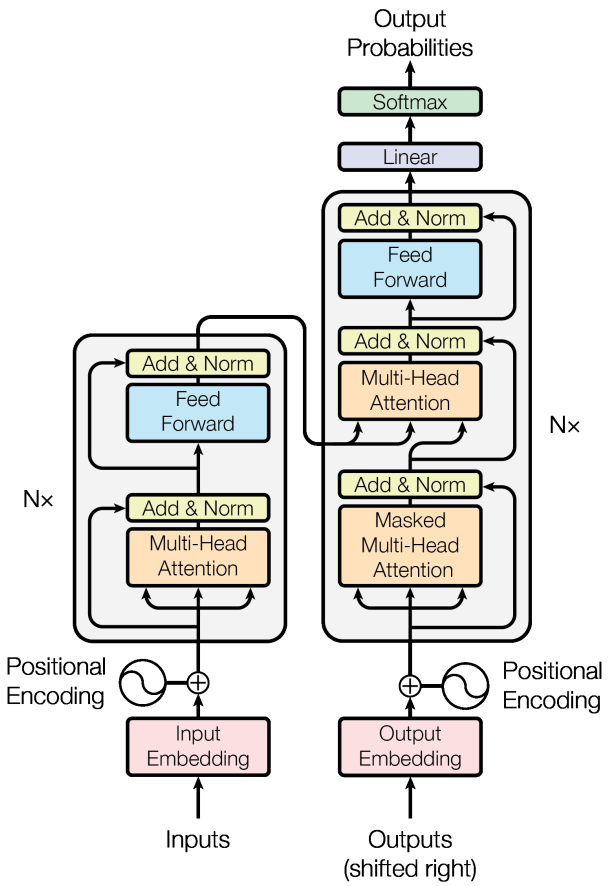

Figura 1. Arquitetura do modelo *Transformer*. (Fonte: [Vaswani *et al*., 2017])

Conforme observado na figura, o *Transformer* segue a arquitetura *Encoder-Decoder* e é composto por blocos idênticos baseados em mecanismos de atenção e redes *feed-forward*.

O *encoder* processa a sequência de entrada e a transforma em representações contínuas ricas em contexto. No artigo original, ele é composto por N camadas idênticas (N=6), em que cada camada combina *Multi-Head Self-Attention* e uma rede *Feed-Forward Position-wise*. Já o *decoder* gera a sequência de saída, uma palavra por vez. Também possui N camadas e inclui *Masked Multi-Head Self-Attention* (para manter a propriedade auto-regressiva), *Encoder-Decoder Attention* e uma rede *Feed-Forward Position-wise*. Como não há recorrência, codificações posicionais são somadas aos *embeddings* para injetar informação de ordem e formar o *positional encoding*. Ao final, a saída do decoder passa por uma transformação linear e uma função *softmax*, produzindo as probabilidades da próxima palavra no vocabulário [Vaswani *et al*., 2017].

Em vez de processar as palavras uma a uma, o *Transformer* as processa simultaneamente na camada de atenção. A auto-atenção (*self-attention*) permite que, ao representar uma palavra, o modelo avalie a relevância das demais e capture relações complexas no texto.

A ausência de recorrência tornou o *Transformer* massivamente paralelizável em hardware moderno, como as GPUs, permitindo o treinamento de modelos muito grandes em um tempo viável. Essa eficiência, combinada com sua capacidade de modelar o contexto, foi o catalisador que possibilitou o surgimento das LLMs [Zhao *et al*., 2025]. Essa arquitetura se tornou a base para praticamente todos os LLMs modernos e, portanto, é central para a geração de conteúdo textual complexo, como os mapas em formato JSON (*JavaScript Object Notation*) propostos neste trabalho.

4 Trabalhos Correlatos

Utilização de regressão para a predição de mapas gerados proceduralmente. O trabalho propõe combinar GPC com Aprendizado de Máquina para personalizar mapas ao estilo do jogador, utilizando uma IA baseada em regressão linear para predizer parâmetros de geração a partir do desempenho e do feedback em partidas anteriores. A solução foi organizada em uma arquitetura cliente-servidor: um Action RPG 2D em Unity atuou como cliente, enquanto o modelo de regressão linear (treinado em Python com Scikit-learn no Google Colab) foi encapsulado em uma API Flask. O jogo coletava métricas de jogabilidade (tempo, mortes, itens) e o parâmetro de diversão ("fun"), enviando-as ao final de cada partida; a API retornava, em JSON, novos parâmetros (p.ex., largura, altura e densidade de inimigos) para orientar o próximo mapa. Embora a integração técnica tenha sido bem-sucedida, o modelo apresentou limitações devido a um *dataset* pequeno (212 amostras) e com baixa variabilidade, levando a predições ruins em casos menos frequentes (como a tendência de predizer sempre "12" para o número máximo de itens). Também se observou uma limitação conceitual: o modelo não distinguia adequadamente avaliações negativas, podendo repetir mapas mal avaliados por não explorar melhor o feedback "fun" [de Oliveira, 2022]. Diferentemente de Oliveira, este trabalho não busca predizer parâmetros para um gerador procedural tradicional, mas sim investigar o próprio

LLM como agente gerador da matriz espacial, combinando geração textual com validação determinística posterior.

Geração Procedural de Mapas para Jogos 2D. Em outra vertente, é apresentado um algoritmo simples para gerar mapas 2D contínuos com topologia orgânica (cavernas, calabouços e ilhas), evitando a estrutura rígida baseada em salas pré-definidas. O processo ocorre em três etapas: (i) crescimento recursivo a partir de uma peça inicial, garantindo conectividade por meio de entradas compatíveis; (ii) validação de tamanho, descartando mapas que ocupem menos de 1/3 da área; e (iii) correção de coesão, com "limpeza" de saídas inválidas via rotação de peças (incluindo pares) e, em último caso, modificação/remoção de conexões problemáticas. O método produz mapas exportáveis (XML) e se destaca pela simplicidade e pela variedade; como limitações, o trabalho aponta o uso de um único tipo de terreno e a necessidade de avaliações mais detalhadas do ponto de vista do jogador, sugerindo combinações de terrenos e integrações com algoritmos genéticos [de Oliveira Barbosa Leite and de Lima, 2015]. Em relação a Leite e Lima, a presente proposta desloca a etapa criativa para um modelo de linguagem guiado por prompt, preservando a validação algorítmica como camada de controle de qualidade.

GPC evolutiva para jogos de plataforma. Dois trabalhos atacam diretamente o problema de garantir jogabilidade em jogos de plataforma. O primeiro propõe um sistema online para um *endless runner* que usa Algoritmo Genético para evoluir, em tempo real, chunks representados por "Colunas de Genes" (GeneColumn), avaliados por uma função de fitness que simula as trajetórias de salto e penaliza saltos impossíveis, buracos intransponíveis, paredes altas e densidades inadequadas de inimigos. Implementado na Godot em C#, o estudo indica boa viabilidade em tempo real com uma população de 50 a 100 indivíduos e baixa mutação (entre $0,5\%$ e 1%), gerando chunks em cerca de 1.5 segundos; por outro lado, a função de fitness é complexa e fortemente acoplada à física, além de controlar apenas aspectos locais (micro-design) [de Pontes and Gomes, 2020].

Ainda nesse eixo, o segundo trabalho propõe um modelo hierárquico de quatro camadas (Componentes, Padrões, Células e Estruturas de Células) para representar ritmo, repetição e conectividade, e um algoritmo *top-down* que decompõe estruturas até componentes ajustáveis. Um avaliador de dificuldade baseado em física estima a dificuldade por "janelas" espaciais e temporais, e alimenta um construtor que usa *hill-climbing* para ajustar parâmetros até atingir uma dificuldade-alvo. O protótipo do construtor foi implementado com sucesso, embora a geração de Estruturas de Células não-lineares ainda estivesse em aberto na época; o texto também ressalta o risco de ótimos locais inerente ao *hill-climbing* [Compton and Mateas, 2021]. Quando comparado a esses dois, este trabalho se diferencia por tratar mapas *top-down* do gênero *dungeon crawler*, nos quais a principal restrição de jogabilidade é a conectividade espacial entre entrada, saída e entidades, validada por busca de caminho com A*.

Gramática de grafos e paralelização para *roguelike*. Por fim, é descrito um motor de GPC baseado em Gramática de Grafos para mapas de *roguelike*, focado em viabilizar a geração em tempo de execução ao enfrentar o gargalo do Isomorfismo de Subgrafos (complexidade exponencial). A solução explora *multithreading* (OpenMP) para buscar, em paralelo, locais de aplicação das regras de produção e aplicar transformações de forma segura. A abordagem foi validada em um jogo exemplo (*Carcere Exire*) com nove regras, e testes mostraram ganhos expressivos: em mapas de 50 salas, o tempo médio caiu de 18,27 s (sequencial) para 2,32 s (16 *threads*), e em mapas de 100 salas, de 36,22 s para 6,75 s. Como limitação, o artigo observa que as regras foram criadas *ad-hoc* para um jogo específico, exigindo esforço adicional para generalização [Souza *et al*., 2019]. Em contraste com Souza et al., a abordagem aqui proposta reduz a dependência de regras explícitas de produção manual, utilizando o LLM para interpretar descrições temáticas em linguagem natural. Embora essa flexibilidade semântica ainda exija uma etapa posterior de validação rígida, pois o LLM isoladamente não garante coerência topológica.

5 Metodologia

Esta seção descreve o modelo conceitual e o pipeline adaptativo propostos para a Geração Procedural de Conteúdo Espacial (GPCE), combinando LLMs e validação algorítmica determinística. O sistema adota uma abordagem híbrida organizada em um ciclo síncrono de três macroetapas: Geração Semântica, Triagem Baseada em Agentes e Execução Gráfica.

5.1 Arquitetura Geral do Pipeline Híbrido

O fluxo operacional é acionado sempre que o ambiente de jogo demanda a instauração de uma nova fase. Em vez de depender de matrizes estáticas ou distribuições puramente estocásticas baseadas em ruídos matemáticos, a arquitetura utiliza o encadeamento de responsabilidades ilustrado no fluxograma da Figura 2.

O processo inicia-se com a leitura dos parâmetros da fase corrente (como dimensões e densidade de entidades) para a montagem de um comando textual direcionado. O LLM atua estritamente na porção criativa da área jogável. Emitida a resposta da IA, o sistema isola o bloco de dados, trata formalmente a matriz e submete o layout ao Agente Validador. Se o mapa for classificado como jogável, a fase é instanciada; caso contrário, o layout é descartado e uma nova iteração de busca é disparada até o limite de 5 tentativas.

5.2 Engenharia de Prompt e Representação Espacial

A Engenharia de Prompt foi modelada sob a premissa de mitigar a volatilidade inerente aos modelos probabilísticos autorregressivos. Para garantir previsibilidade estrutural sem cercear a distribuição semântica dos elementos, o prompt enviado à IA é dividido em cinco blocos diretivos:

1. **Atribuição de Papel e Formatação Estrita**: O modelo é instruído, logo na primeira linha, a atuar exclusivamente como um gerador procedural de mapas para um jogo *dungeon crawler 2D* em visão *top-down*. Para viabilizar a comunicação direta com o motor de jogo, aplica-se uma restrição: a saída deve ser estritamente um objeto JSON codificado, sendo proibida a geração de textos conversacionais ou formatação *markdown*.
2. **Direcionamento Temático (*Map Description*)**: Injeta-se um bloco descritivo contendo o tema desejável do

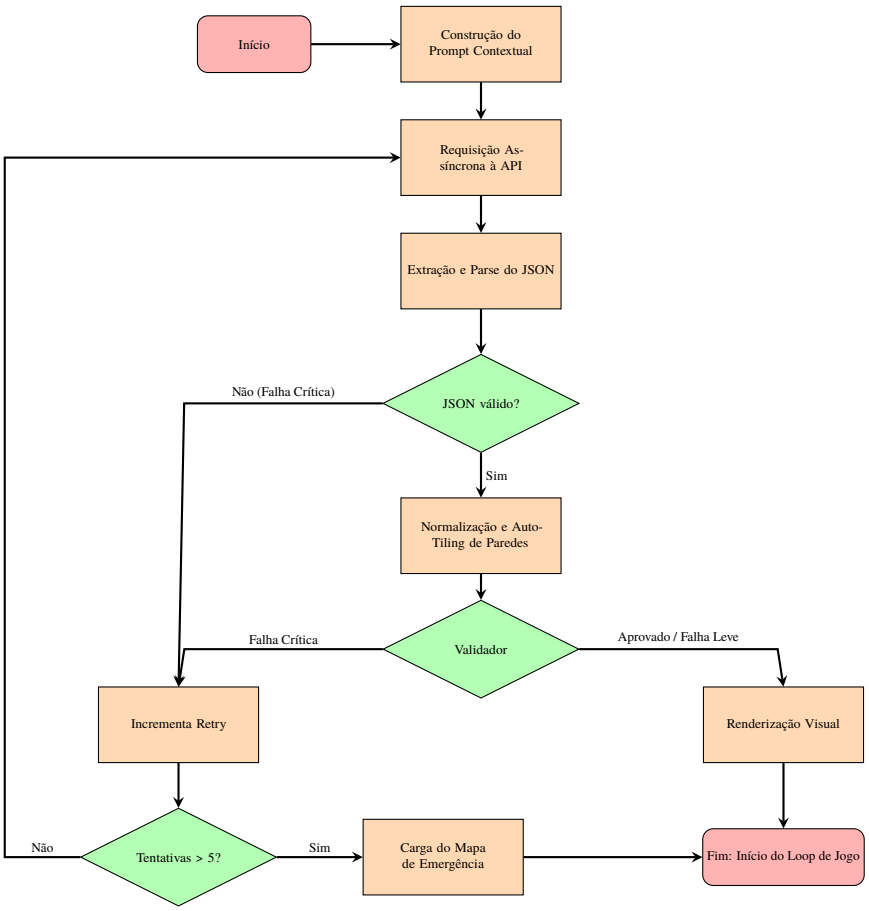

Figura 2. Fluxograma do pipeline de geração, validação e contingência estrutural

nível (variável `theme`), que atua como âncora semântica para a construção de formas orgânicas e para o posicionamento de decorações e entidades de modo coerente com a atmosfera fornecida.

3. **Gramática ASCII e Legenda de Entidades**: Fornece-se um dicionário de mapeamento de caracteres, onde cada símbolo está atrelado a um nó ou cena do jogo. O ambiente vazio é definido por `-`, enquanto áreas jogáveis e corredores são representados por `.`. Exige-se exatamente uma entrada (`P`) e uma saída (`S`), além de parametrizar as densidades para baús (`b, B`), inimigos (`e, E, V`) e obstáculos cenográficos (`c, r, p`).
4. **Regras Estruturais e Topológicas (*Map Rules*)**: Um subsistema de cinco regras explícitas governa a validade espacial do labirinto. O LLM deve respeitar dimensões estritas de colunas (`width`) e linhas (`height`) e posicionar `P` e `S` em extremidades opostas da matriz, garantindo um caminho livre e contínuo de caracteres de chão (`.`) conectando entrada e saída.
5. **Demonstração por Amostragem (*One-Shot Prompting*)**: O comando é finalizado com um exemplo de formatação JSON preenchido com uma miniatura 8x8, ancorando o padrão estrutural esperado e reforçando que o arranjo final da matriz deve replicar o formato, expandido para as dimensões solicitadas.

O prompt utilizado nos experimentos foi mantido constante em sua estrutura geral, variando apenas os parâmetros específicos de cada fase, como tema, largura, altura e quantidade esperada de entidades. A Figura 3 apresenta o modelo de prompt empregado para orientar a geração da matriz JSON pelas LLMs.

O JSON encapsula a matriz bidimensional em uma propriedade identificadora específica, eliminando a necessidade de rotinas complexas de processamento de linguagem natural no motor de jogo e permitindo o mapeamento direto das linhas ASCII para arrays nativos em GDScript.

As instruções foram redigidas predominantemente em forma afirmativa e mandatória, evitando restrições negativas, pois estudos demonstram que modelos de linguagem apresentam dificuldades sistemáticas na interpretação de instruções contendo negação [Jang *et al.*, 2023; Truong *et al.*, 2023]. Durante o processamento, o conceito negado ainda precisa ser representado internamente, o que pode aumentar a ocorrência de respostas inconsistentes ou a inclusão inadvertida de elementos proibidos. Assim, recomenda-se especificar explicitamente o comportamento desejado em vez de apenas proibir comportamentos indesejados [Zhou *et al.*, 2026].

5.3 Algoritmo do Agente Validador Autônomo e Ciclo de Resiliência

Como as LLMs operam por meio da predição estatística sequencial de tokens e não possuem cognição geométrica intrínseca, a saída bruta não é confiável para uso imediato no loop de gameplay. O Agente Validador soluciona essa lacuna por meio de uma triagem sequencial em três fases analíticas:

```
You are a procedural map generator for a 2D top-down dungeon crawler game.
Your ONLY task is to generate a dungeon map and output it strictly as an encoded JSON object. No markdown, no
 conversational text.

MAP DESCRIPTION (THEME):
{theme}

CHARACTER LEGEND:
'-' : Empty void outside the map;
'.' : Floor (Playable path);
'P' : Player spawn point entrance (place only 1);
'S' : Exit door (place only 1);
'b' : Common chest (place {common_chest});
'B' : Rare chest (place {rare_chest});
'e', 'E', 'V' : Enemies: easy, medium, hard (place at least {total_enemies} total);
'c', 'r', 'p' : Decorations or obstacles: crates, rocks, pillars.

MAP RULES:
1. Dimensions: The map must be {width} columns (width) by {height} rows (height).
2. Playable Area: Create an organically shaped area (connected rooms and corridors) using floors ('.'), the
 player ('P'), exit ('S'), and objects.
3. Outer Void: Use '-' to represent the empty space outside the playable area.
4. Path: Ensure that exists an player entrance ('P') and an the exit ('S'). Ensure that a clear path of floors
 ('.') connecting the entrance and the exit, and ensure the entrance and exit are on opposite sides of the map.
5. Distribution: Place objects naturally according to the MAP DESCRIPTION.

REQUIRED OUTPUT JSON FORMAT EXAMPLE (The example below is a 8x8 miniature. YOUR output MUST be {width}x{height}):
{example}
```

Figura 3. Prompt utilizado para a geração procedural de mapas.

5.4 Validação Estrutural e Sintática

A primeira barreira avalia a integridade do arquivo e as proporções da malha recebida. O sistema verifica se o texto extraído constitui um objeto JSON íntegro e se as dimensões reais da matriz gerada estão dentro de uma tolerância paramétrica de ±30% em relação aos valores solicitados. Se o layout falhar nesta etapa, ele recebe o rótulo de Falha Crítica (`CRITICAL_FAULT`).

A tolerância paramétrica de ±30% foi adotada como decisão heurística de projeto, com o objetivo de equilibrar rigidez estrutural e resiliência operacional. Como as LLMs frequentemente geram linhas com pequenas variações de comprimento, uma exigência absolutamente exata de largura e altura descartaria mapas potencialmente aproveitáveis, ainda que corrigíveis pelo processo posterior de normalização. Por outro lado, tolerâncias excessivamente amplas permitiriam a aceitação de matrizes incompatíveis com a fase solicitada, distorcendo a escala do nível e comprometendo a comparação experimental. Assim, o intervalo de ±30% foi utilizado como margem operacional intermediária: suficiente para acomodar desvios moderados da geração textual, mas restritivo o bastante para rejeitar mapas estruturalmente distantes da configuração solicitada. Essa escolha não foi derivada de otimização estatística prévia, sendo tratada como parâmetro de projeto a ser refinado em trabalhos futuros.

5.5 Validação Topológica de Conectividade (Algoritmo *A**)

Esta etapa constitui o núcleo lógico de controle de qualidade do agente. A matriz de caracteres é convertida em um grafo de navegação bidimensional ortogonal, com restrição absoluta de movimentos diagonais. O validador executa varreduras com o algoritmo de busca de caminho *A**, utilizando a heurística de distância *Manhattan*. Para assegurar a solucionabilidade real do nível, o algoritmo implementa regras condicionais de colisão abstrata:

- **Validação da Rota de Saída** (`P` $\rightarrow$ `S`): O agente testa a existência de ao menos uma rota viável entre o ponto inicial do jogador (P) e algum ponto de saída do nível (S). Durante a varredura da matriz extraída do JSON, caso mais de um ponto P seja encontrado, o sistema adota como posição inicial efetiva o último P identificado, garantindo que haja uma única coordenada de início para a simulação de navegação. Já para as saídas, todos os pontos S encontrados são armazenados como destinos possíveis. Assim, em vez de exigir caminho até uma saída específica, o validador verifica se existe ao menos um caminho entre a entrada efetivamente escolhida e qualquer uma das saídas disponíveis. Durante essa busca, os baús de tesouro (b, B) são parametrizados temporariamente como obstáculos sólidos intransponíveis, impedindo que o algoritmo aprove rotas cujo acesso dependa da travessia física de um item sólido.
- **Validação de Acessibilidade de Entidades**: O agente verifica se os inimigos e os baús espalhados estão fisicamente acessíveis a partir do ponto inicial. Nesta varredura secundária, o ponto de saída `S` é mascarado como um bloqueio sólido. Esse procedimento é indispensável para evitar o falso positivo em que um item é posicionado atrás da porta de saída; teoricamente a rota existiria, mas na prática o jogador encerraria a fase de forma prematura ao tocar em `S` antes de interagir com a entidade avaliada.

Se o algoritmo falhar em traçar um caminho para a saída, ou se os pontos essenciais `P` e `S` estiverem ausentes, o layout

é sentenciado como Falha Crítica.

5.6 Validação de Regras do Prompt e Classificação Categórica

Caso o mapa possua conectividade funcional, o agente analisa critérios secundários qualitativos (contagem de entidades esperadas vs. geradas e identificação de caracteres alucinado estranhos à legenda). Se houver discrepâncias de baús ou inimigos, o layout é aprovado com restrições e classificado como Falha Leve (`LIGHT_FAULT`), sendo aceito para renderização final por manter a jogabilidade intacta. O resultado final se consolida em três estados categóricos mutuamente exclusivos:

$$\text{Resultado} = \begin{cases} \texttt{APPROVED}, & \text{se o mapa for solucionável e cumprir todas as metas de entidades.} \\ \texttt{LIGHT\_FAULT}, & \text{se o mapa for solucionável, mas divergir nas metas qualitativas.} \\ \texttt{CRITICAL\_FAULT}, & \text{se o mapa for insolúvel ou apresentar anomalias sintáticas.} \end{cases} \tag{2}$$

É importante destacar que apenas mapas classificados como `CRITICAL_FAULT` disparam uma nova tentativa de geração pelo LLM. Mapas classificados como `LIGHT_FAULT` não retornam ao ciclo de retentativa, pois são considerados funcionalmente jogáveis: possuem conectividade válida, entrada e saída acessíveis e não bloqueiam a progressão do jogador. Nesses casos, as divergências detectadas dizem respeito a critérios secundários de aderência ao prompt, principalmente diferenças na quantidade de entidades geradas em relação às quantidades solicitadas.

Sempre que a classificação `CRITICAL_FAULT` é emitida, o sistema incrementa o contador de controle local (*current_retries*). Se o contador estiver contido no intervalo de segurança (≤ 5), o pipeline reconfigura o estado da chamada e solicita uma nova matriz estrutural ao LLM. Ao atingir o teto crítico de tentativas sem obter um layout aprovado, o motor de jogo aborta as chamadas de inferência e carrega o mapa de contingência local correspondente à fase atual, blindando o loop de gameplay de falhas catastróficas consecutivas. Todo o histórico cronometrado de geração, tempos de validação e taxas de erro é compilado de forma transparente em um arquivo *CSV* para persistência e auditoria científica posterior.

A classificação categórica também define o grau de reaproveitamento possível do mapa gerado. Mapas classificados como *APPROVED* podem ser utilizados integralmente. Mapas classificados como *LIGHT_FAULT* são considerados reaproveitáveis para gameplay, pois preservam conectividade, entrada, saída e navegabilidade, embora apresentem desvios em metas secundárias, como quantidade de inimigos ou baús. Já mapas classificados como *CRITICAL_FAULT* não são utilizados diretamente, pois podem conter JSON inválido, dimensões incompatíveis, ausência de pontos essenciais ou bloqueios topológicos capazes de gerar soft-locks. Ainda assim, partes desses mapas, como padrões visuais, distribuição de salas ou elementos temáticos, poderiam ser reaproveitadas em trabalhos futuros por meio de um módulo de reparo procedural, desde que submetidas a nova validação determinística.

6 Implementação

A arquitetura teórica do sistema foi materializada na forma de um protótipo funcional utilizando a *Godot Engine* (versão 4.6), com toda a lógica desenvolvida em GDScript. A implementação foi segregada em três módulos principais que operam em cascata: comunicação e inferência, pós-processamento de matrizes e validação de navegabilidade.

6.1 Comunicação e Geração Assíncrona

O módulo principal de controle do nível (`level_generator.gd`) gerencia a interface com as APIs de Inteligência Artificial. Para evitar o congelamento do *loop* principal de renderização enquanto aguarda o processamento do LLM, a comunicação foi implementada de maneira assíncrona por meio de requisições HTTP (`HTTPRequest`). O sistema permite a alternância entre inferência local (via Ollama) e modelos em nuvem (API Google Gemini).

Na montagem do corpo da requisição (*payload*), o parâmetro de temperatura (*temperature*) foi fixado em 1.0, mantendo a configuração padrão dos modelos utilizados e buscando preservar a variabilidade criativa na geração dos mapas. Em termos práticos, valores mais altos de temperatura tendem a aumentar a diversidade das respostas, favorecendo layouts menos repetitivos e maior variação estética, mas também podem ampliar a instabilidade sintática e a divergência em relação às regras do prompt. Valores mais baixos, por sua vez, poderiam tornar as respostas mais determinísticas e possivelmente mais aderentes ao formato solicitado, porém com risco de reduzir a diversidade espacial e temática dos mapas. Como este trabalho não realizou um estudo de degradação variando a temperatura, os resultados devem ser interpretados como representativos do comportamento dos modelos sob uma configuração padrão e criativamente permissiva.

Para lidar com falhas inerentes a essa configuração permissiva, o controlador implementa um ciclo recursivo de chamadas limitado pela constante `MAX_RETRIES = 5`. Esse limite foi definido como um parâmetro heurístico de orçamento computacional, equilibrando resiliência e custo de inferência. Um número muito baixo de tentativas, como 1 ou 2, reduziria significativamente a chance de recuperação diante da alta taxa de falhas críticas observada nas saídas brutas das LLMs. Por outro lado, um número muito elevado, como 10 ou mais tentativas, poderia tornar a geração impraticável em termos de latência, especialmente nos modelos locais, cujos tempos de inferência são substancialmente maiores. Assim, o limite de cinco tentativas foi adotado como compromisso operacional entre tolerância a falhas e tempo máximo aceitável de espera. Caso o limite se esgote, o sistema aciona um *fallback* local por meio da função *_load_fallback_map*, lendo um arquivo JSON pré-fabricado para garantir a estabilidade do jogo.

6.2 Pós-processamento e Geração Automática de Paredes

Uma decisão de projeto fundamental foi abstrair do LLM a responsabilidade de delimitar fisicamente os cenários. Por operarem de forma sequencial e textual, modelos autorregressivos não possuem percepção espacial e falham em garantir

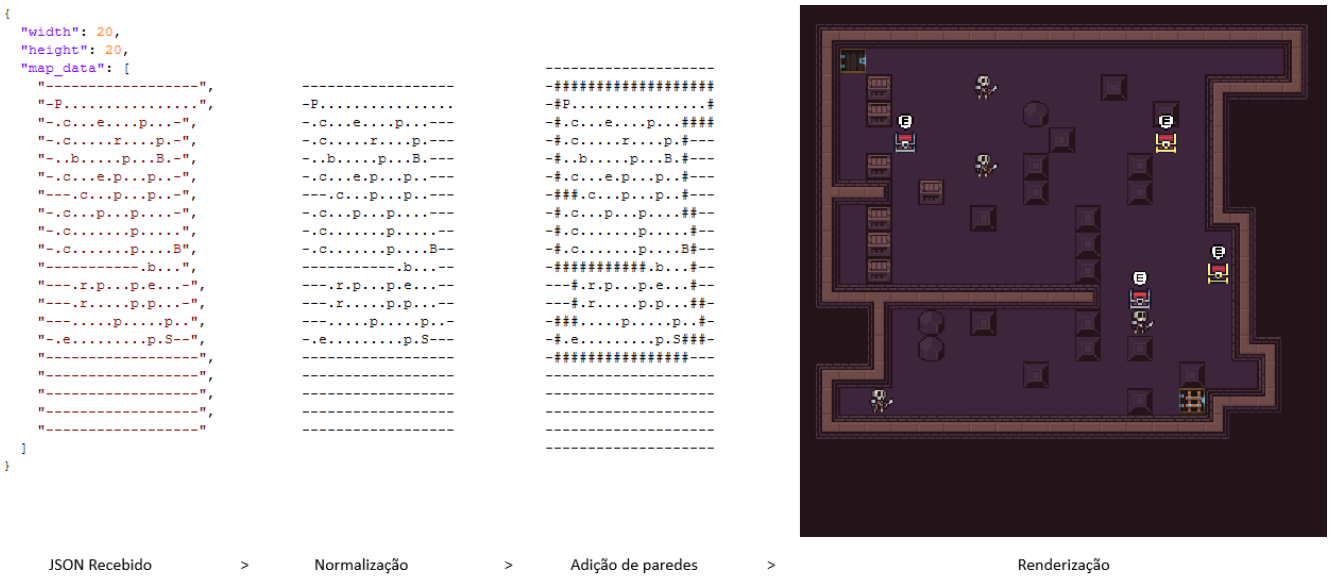

Figura 4. Evolução processamento JSON

fechamentos topológicos exatos. Assim, a IA foi instruída a gerar apenas o espaço jogável, delegando a construção das paredes ao módulo utilitário (`map_utils.gd`), que processa a matriz em duas etapas:

- **Normalização**: Como a IA tende a produzir linhas com larguras irregulares, o algoritmo identifica a linha mais longa e preenche as menores à direita com espaços vazios (`-`), garantindo uma grade retangular simétrica (Figura 4).
- ***Auto-tiling* Analítico**: Após adicionar uma borda de segurança ao mapa, o script itera sobre cada espaço vazio (`-`). Se o espaço possuir um caractere de área jogável em qualquer um de seus oito vizinhos (`Vector2i`), ele é convertido em uma parede sólida (`#`). Esse processo determinístico garante um fechamento hermético do cenário, mitiga erros geométricos e economiza tokens de processamento.

6.3 O Algoritmo de Validação Espacial

Para a validação topológica, o módulo `map_validator.gd` converte a matriz textual em uma grade de navegação abstrata ao instanciar a classe nativa `AStarGrid2D`. As movimentações são estritamente restritas ao eixo ortogonal (`DIAGONAL_MODE_NEVER`). A malha é dinamicamente populada e os obstáculos variam conforme a varredura em execução:

- **Teste de Solucionabilidade**: Ao calcular a viabilidade da rota entre a Entrada (`P`) e qualquer Saída (`S`), o script injeta um *array* de caracteres que determina os obstáculos. Além das paredes e do vácuo exterior, todos os baús e elementos de decoração (`c, r, p, b, B`) são repassados ao método `set_point_solid` como instâncias intransponíveis.
- **Teste de Acessibilidade de Itens**: Durante a varredura para validar se a entidade inimiga ou o espólio é alcançável, o ponto de destino se altera; neste cenário, a porta de saída (`S`) é reclassificada e injetada no *array* de itens sólidos, prevenindo bloqueios conceituais de colisão cruzada.

6.4 Renderização Dinâmica e Telemetria

Após a aprovação do Agente Validador, a matriz ASCII é traduzida em representações gráficas através do nó `TileMapLayer`. O ambiente é pintado com o uso da função `set_cells_terrain_connect`, nativa da engine, que aplica regras automáticas de conectividade e seleção de *sprites* (*Auto-Tiling*) baseadas nos vizinhos para renderizar as paredes suavemente.

Simultaneamente, visando a coleta quantitativa de dados para análise, os parâmetros da transação (ID do modelo, tempo de resposta em milissegundos, quantidade de entidades requeridas e geradas, bem como a classificação final e os motivos das falhas) são extraídos pelo validador. O método `_save_to_csv` persiste as informações em um documento CSV no diretório do usuário (`user://`) para posterior tabulação científica.

6.5 Disponibilidade do Protótipo

O código-fonte do protótipo desenvolvido neste trabalho, incluindo os módulos de geração, validação, pós-processamento e persistência dos resultados experimentais, está disponível publicamente em repositório GitHub: `https://github.com/elciofagundesboin/2D-Map-Generation-with-LLMs-and-Agent-Based-Validation`.

6.6 Exemplos

A seguir, alguns exemplos de mapas obtidos a partir desta implementação. Na figura 5 são exibidos exemplos de mapas com boas características visuais: todas as entidades e itens estão ao alcance do jogador, e o caminho da Entrada até a Saída é orgânico. Enquanto nos mapas da figura 6, apesar de ser possível alcançar a Saída, não é possível acessar algumas regiões, bem como os baús e inimigos.

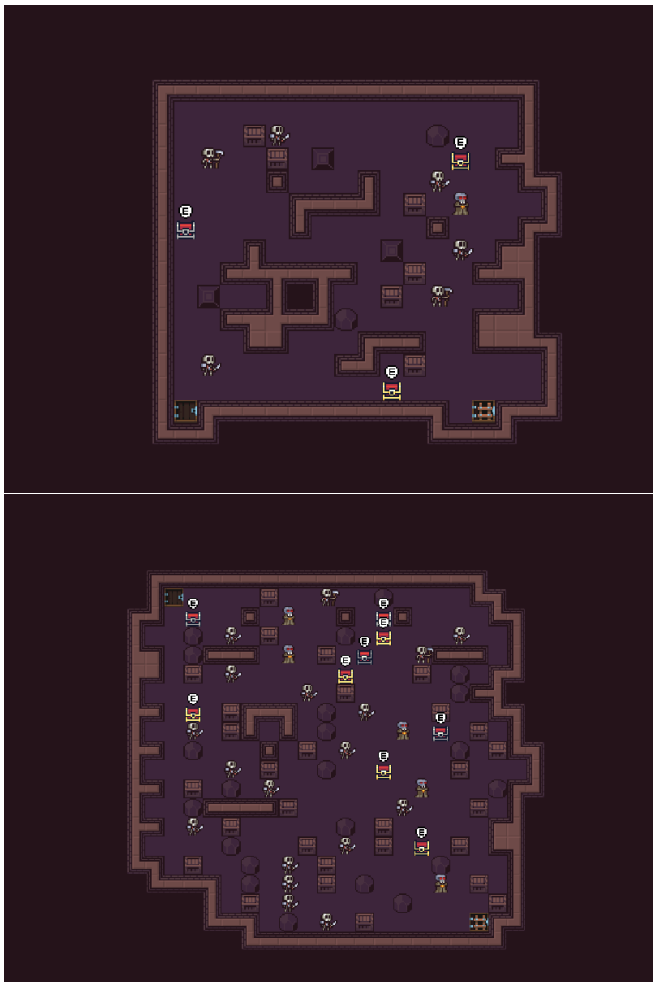

Figura 5. Mapas aprovados gerados.

Figura 6. Mapas reprovados gerados.

7 Resultados

7.1 Metodologia de Coleta e Ambiente de Testes

Para avaliar a viabilidade e a robustez do pipeline de geração e validação, conduziu-se uma bateria de testes simulando campanhas completas do jogo, compreendendo 5 diferentes temas, complexidade geométrica crescente (15x15, 20x20 e 30x30) e densidade crescente de entidades. O limite estrito de até 5 tentativas sequenciais de geração por mapa foi mantido. O experimento contrastou três LLMs: dois modelos de execução local, sendo Gemma 4 (com 4 bilhões de parâmetros e pesando 9,6 GB); e Ministral 3 (com 14 bilhões de parâmetros e pesando 9,1 GB); e um modelo baseado em nuvem via API (Gemini 3.1 Flash Lite). Para os modelos locais, foram feitas 5 campanhas completas - com cada conjunto de configuração (dimensão vs entidades) - e para o modelo em nuvem, 2 campanhas completas. Onde foram gerados um total de 1885 arquivos em 108 campanhas completas.

Os testes de inferência local foram executados integralmente em hardware comercial, utilizando uma placa de vídeo NVIDIA RTX 3060 equipada com 12 GB de VRAM. Devido às restrições de rate limit da API em nuvem, o modelo Gemini operou com um volume amostral menor em relação aos modelos locais, mantendo, contudo, a significância estatística para fins comparativos.

7.2 Procedimentos de Cálculo Estatístico

A análise quantitativa foi realizada a partir dos arquivos CSV gerados automaticamente pelo protótipo, nos quais cada linha representa uma tentativa de geração de mapa realizada por um determinado modelo, com informações sobre nível, dimensões solicitadas, tempo de geração, número de tentativa, entidades esperadas e geradas, status final de validação e motivo de falha, quando aplicável.

Nas Figuras 7, 8 e 9, as taxas percentuais foram calculadas por frequência relativa. Na Figura 7, os dados foram agrupados por modelo, contabilizando-se o total de mapas gerados por cada LLM e a quantidade de ocorrências de cada status de validação (`APPROVED`, `LIGHT_FAULT` e `CRITICAL_FAULT`). A porcentagem de cada status foi calculada pela razão entre a contagem daquele status e o total de gerações do respectivo modelo:

$$P(status) = \frac{n_{status}}{n_{total\_modelo}} \times 100$$

Na Figura 8, o mesmo cálculo foi aplicado após segmentação simultânea por modelo e nível da fase, de modo que cada porcentagem representa a proporção de determinado status dentro de uma combinação específica de modelo e nível. Na Figura 9, a segmentação foi feita por modelo e dimensão do mapa, a partir da variável derivada *Map_Size*, formada pela concatenação da largura e da altura da matriz, como 15x15, 20x20 e 30x30.

Na Figura 10, o tempo bruto de geração, originalmente

registrado em milissegundos, foi convertido para segundos pela divisão por 1000. A distribuição foi representada por boxplots, que sintetizam a mediana, os quartis Q1 e Q3, o intervalo interquartil, os limites de dispersão definidos por 1,5 vez o intervalo interquartil e os valores discrepantes plotados como *outliers*. Assim, o gráfico não representa apenas uma média de tempo, mas a variabilidade da latência de geração em cada combinação de modelo e nível.

Na Figura 11, foi utilizada a média aritmética simples para comparar a quantidade de inimigos esperada no prompt e a quantidade efetivamente gerada. Os dados foram agrupados por modelo e nível, calculando-se, para cada grupo, a média de *Expected_Enemies* e de *Generated_Enemies*.

Na Figura 12, consideraram-se apenas as tentativas classificadas como `CRITICAL_FAULT`. Para cada modelo, foram contabilizadas as ocorrências de cada motivo de falha crítica registrado em *Critical_Reason*. A porcentagem de cada motivo foi calculada em relação ao total de falhas críticas daquele mesmo modelo, e não em relação ao total geral de mapas gerados:

$$P(motivo) = \frac{n_{motivo}}{n_{falhas\_criticas\_modelo}} \times 100$$

Na Figura 13, foram filtrados apenas os mapas considerados jogáveis, isto é, aqueles classificados como `APPROVED` ou `LIGHT_FAULT`. Em seguida, foi realizada uma contagem absoluta de ocorrências por número de tentativa, de 1 a 5, permitindo observar em qual iteração o pipeline obteve mapas aceitáveis para uso.

Nas Figuras 14, 15 e 16, referentes à avaliação humana, as respostas em escala Likert foram tratadas numericamente de 1 a 5, em que 1 representa ausência de semelhança com o tema solicitado e 5 representa excelente representação temática. Para a média global de coesão da Figura 14, foi utilizada média ponderada, calculada pela soma dos votos em cada alternativa multiplicados por seu respectivo peso, dividida pelo total absoluto de votos:

$$\bar{x}_{ponderada} = \frac{\sum_{i=1}^{5}(v_i \times w_i)}{\sum_{i=1}^{5} v_i}$$

onde v_i representa o número de votos na categoria i e w_i representa o peso associado à categoria. Na Figura 15, cada bloco da barra empilhada representa a proporção percentual de votos de uma categoria Likert em relação ao total de votos recebidos pelo modelo. Por fim, na Figura 16, foi calculada inicialmente uma média ponderada por linha de avaliação e, posteriormente, a média aritmética dessas médias para cada combinação de modelo e nível/tema.

7.3 Taxa de Sucesso e Resiliência Estrutural

A análise primária buscou responder à questão da confiabilidade autônoma dos modelos em gerar conteúdo estritamente jogável. A Figura 7 revela que a saída bruta das LLMs é altamente instável para fins de Level Design, sem intervenção. O modelo Gemma4 apresentou a maior taxa de falhas críticas (86,0%), seguido pelo Gemini-3.1-flash-lite (80,6%) e pelo Ministral-3:14b (77,0%). As aprovações perfeitas (`APPROVED`), onde o mapa era topologicamente viável e as metas de entidades foram exatamente cumpridas, foram estatisticamente marginais, atingindo um pico de apenas 2,5% no Gemini.

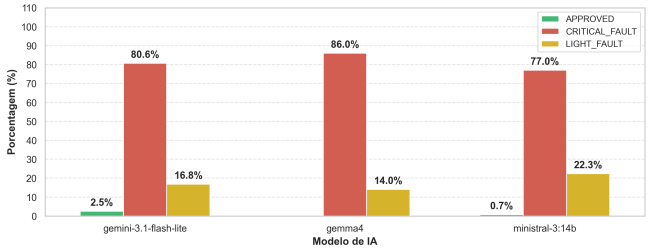

Figura 7. Taxa de Sucesso Global

A fragilidade espacial das IAs torna-se mais evidente ao cruzar a degradação de performance com o aumento da complexidade. A Figura 8 demonstra que as falhas críticas escalam progressivamente nos níveis mais avançados, atingindo picos de 94,2% no nível 3 para o modelo Gemini. Paralelamente, a Figura 9 ilustra o impacto da expansão da área geométrica: ao escalar a matriz de 15x15 para 30x30, o modelo Gemma4 saltou de 75,5% para 93,9% de rejeição crítica, indicando uma diluição drástica da capacidade de atenção do modelo (*attention mechanism*) quando forçado a manter o contexto estrutural sobre longas cadeias de tokens.

7.4 Desempenho e Latência

A viabilidade da integração da GPCE em tempo de execução depende intimamente do tempo de resposta. A Figura 10 apresenta o contraste entre inferência local e processamento em nuvem. No hardware utilizado, os modelos Gemma4 e Ministral-3:14b mantiveram medianas operacionais oscilando entre 40 e 50 segundos por mapa. Em contrapartida, o modelo Gemini resolveu a matriz frequentemente em menos de 10 segundos. Estes dados sugerem que, com o hardware atual, o uso de SLMs (Small Language Models) locais para jogos em tempo real exige o mascaramento da latência através de telas de carregamento dinâmicas ou geração assíncrona em segundo plano.

7.5 Adesão ao Prompt e o Fenômeno da Contagem

A precisão semântica e a obediência às restrições quantitativas (Tabela 1 e Figura 11) expõem limitações intrínsecas à arquitetura dos Transformers. A Tabela de Caracteres Inválidos documenta que o modelo Gemma4 sofreu altos índices de alucinação sintática (presentes em 31,1% dos seus mapas gerados), injetando caracteres que não constavam na legenda fornecida no prompt. O modelo Gemini demonstrou a maior aderência sintática, alucinando caracteres em apenas 1,0% dos casos.

Tabela 1. Ocorrência de caracteres inválidos nos mapas gerados.

Modelo	Soma de Caracteres	Mapas c/ Alucinação	% do Total
gemini-3.1-flash-lite	3	3	1,0%
gemma4	4385	268	31,1%
ministral-3:14b	180	43	6,1%

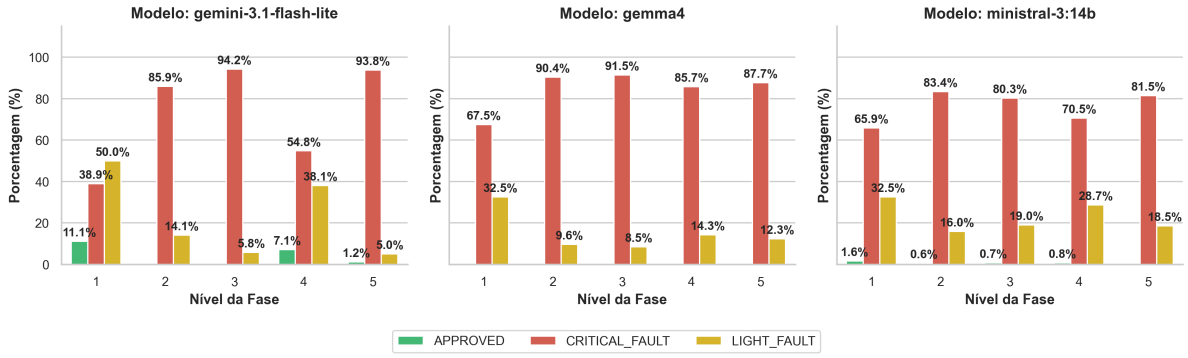

Figura 8. Distribuição de Status por Nível e Modelo

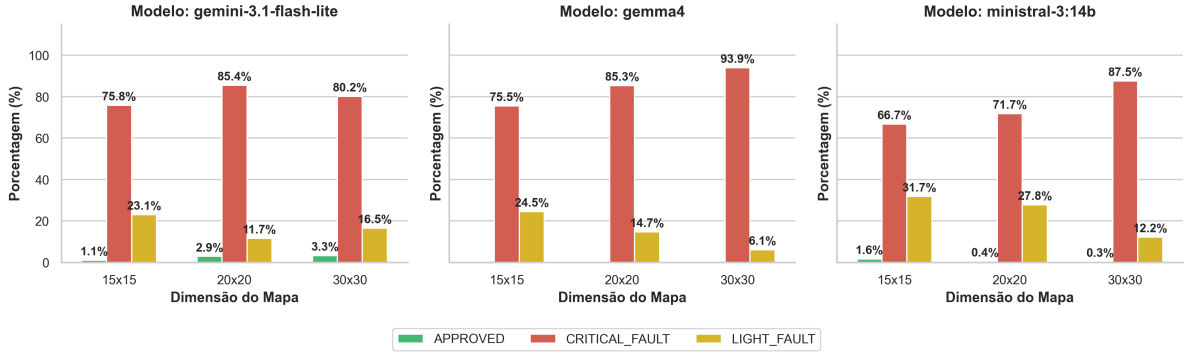

Figura 9. Distribuição de Status por Tamanho do Mapa

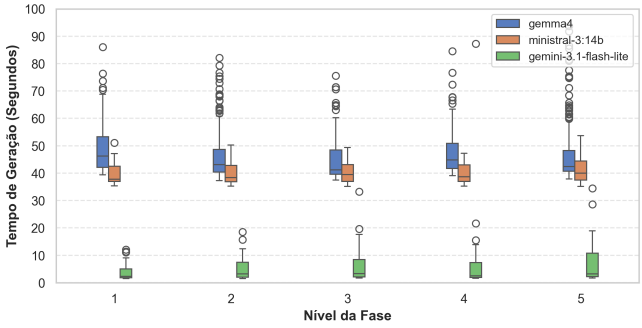

Figura 10. Distribuição do Tempo de Geração por Nível

A Figura 11 evidencia o conhecido "Problema da Contagem" em LLMs. À medida que o nível exigia uma densidade maior de entidades (linha azul do alvo), as médias geradas (barras vermelhas) mostram desvios significativos, com o modelo Gemini superpovoando a fase 5 (média de 10 inimigos onde a meta era 6), enquanto os demais flutuavam erraticamente. Isso ocorre porque as LLMs processam palavras como sub-unidades (tokens) e calculam probabilidades sintáticas, não possuindo um registrador aritmético nativo para contar instâncias já geradas na matriz.

7.6 Coerência Topológica e a "Cegueira Espacial"

O insight topológico constitui o fenômeno mais crítico isolado pela experimentação, aqui cunhado como "Cegueira Espacial". A Tabela 2 de Entidades Inalcançáveis revela que, embora as LLMs instanciem os baús e inimigos na matriz JSON, em cerca de 5% a 8% do volume total, essas entidades foram cimentadas em enclaves fechados, isoladas da área principal de gameplay.

Tabela 2. Ocorrência de bloqueios topológicos nos mapas gerados.

Modelo	Soma Entidades Presas	Mapas c/ Bloqueio	% do Total
gemini-3.1-flash-lite	181	27	8,6%
gemma4	236	56	6,5%
ministral-3:14b	264	41	5,8%

Essa anomalia é aprofundada na proporção de erros apresentada na Figura 12. Para o modelo Gemini, impressionantes 90,6% de suas falhas críticas derivaram da incapacidade de construir um caminho conectando o Jogador (P) à Saída (S). O modelo Ministral-3:14b, embora superior na conectividade, falhou em gerar os próprios caracteres P e S em 53,6% das falhas críticas. Estes resultados comprovam que a inteligência artificial apreende o "o quê" (a semântica e a sintaxe), mas o processamento unidimensional de tokens em texto plano priva o modelo de um raciocínio bidimensional coordenado, impedindo que a IA projete alinhamentos de paredes múltiplas linhas acima ou abaixo para garantir rotas abertas de *pathfinding*.

7.7 A Eficácia do Agente Validador Autônomo

Diante da extrema volatilidade das LLMs mostrada nas seções anteriores, a contribuição central deste sistema reside no pipeline de mitigação. A Figura 13 prova estatisticamente a resiliência do algoritmo. A triagem oculta promovida pelo algoritmo $A*$ atua como uma peneira determinística. Embora grande parte dos mapas jogáveis (classificados como `LIGHT_FAULT` ou `APPROVED`) sejam obtidos na primeira tentativa, um montante cumulativo massivo foi resgatado nas tentativas 2, 3, 4 e 5. Sem este agente validador executando

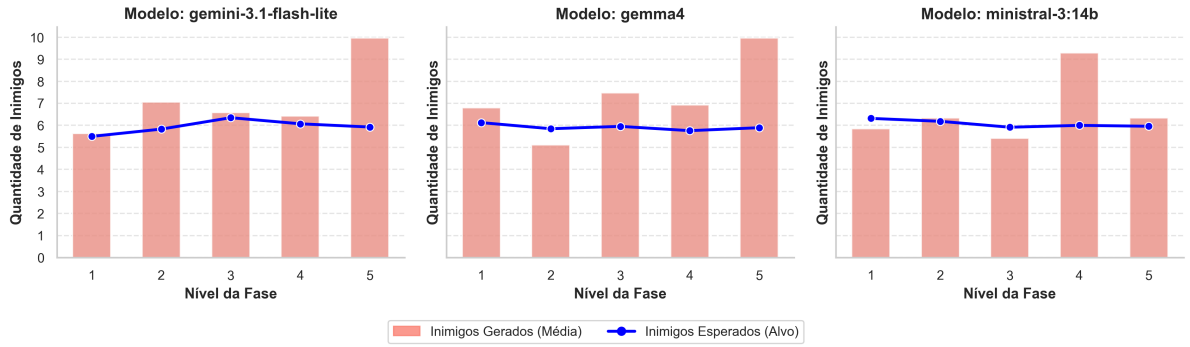

Figura 11. Inimigos Gerados vs Esperados por Nível

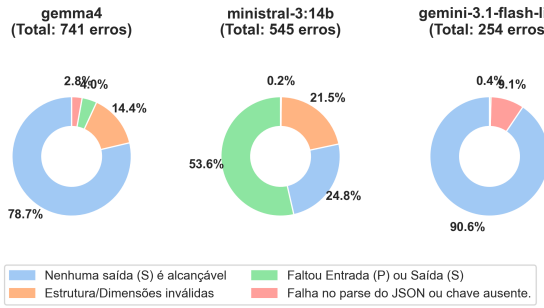

Figura 12. Proporção de Motivos de Falhas Críticas

repetições controladas e ocultas, o jogador seria invariavelmente exposto a layouts que travam o progresso do jogo (*softlocks*), demonstrando que LLMs não podem operar sistemas espaciais isoladamente.

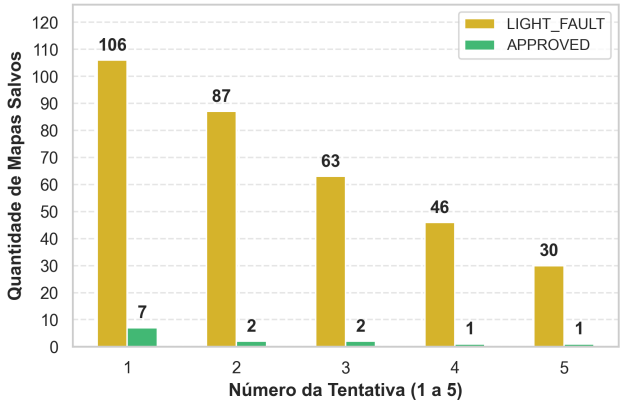

Figura 13. Mapas Jogáveis Salvos por Tentativa

7.8 Avaliação Qualitativa e Percepção Humana

Para avaliar a adequação atmosférica e visual dos cenários gerados, conduziu-se uma avaliação qualitativa humana por meio de formulário estruturado em Escala Likert de cinco pontos, submetido a 13 avaliadores. O questionário foi composto por 25 perguntas, sendo 5 perguntas para cada tema/nível avaliado. Cada pergunta apresentava uma imagem renderizada de um mapa gerado pelo sistema e solicitava que o avaliador indicasse o grau de semelhança entre o mapa exibido e o tema solicitado no prompt.

As alternativas de resposta foram: (1) *"Nenhuma semelhança com o tema solicitado"*; (2) *"Leve semelhança, mas a estrutura predominante é caótica"*; (3) *"Neutro (tem características, mas poderia ser qualquer outra coisa)"*; (4) *"Boa semelhança (representa bem alguns aspectos)"*; e (5) *"Excelente representação do tema"*. A avaliação foi conduzida de forma cega em relação aos modelos: os avaliadores não receberam informação sobre qual LLM havia gerado cada mapa, e as imagens foram apresentadas em ordem embaralhada no formulário, a fim de reduzir vieses de comparação direta entre modelos.

Para cada tema, foram selecionadas cinco imagens: duas geradas pelo modelo Gemma4, duas pelo modelo Ministral-3:14b e uma pelo modelo Gemini-3.1-flash-lite. A diferença na quantidade de imagens do Gemini decorre do menor volume amostral desse modelo, condicionado pelas limitações de uso da API em nuvem durante os experimentos.

Tabela 3. Distribuição quantitativa de votos na escala por modelo e pergunta.

			Escala Likert (1-5)				
Pergunta	**Nível/Tema**	**Modelo**	**1**	**2**	**3**	**4**	**5**
1	1	ministral-3:14b	2		8		3
2	1	gemini-3.1-flash-lite	7	1			5
3	1	ministral-3:14b	2		7	2	2
4	1	gemma4		1	8	2	2
5	1	gemma4	3		8	1	1
6	2	ministral-3:14b	1		3	9	
7	2	gemma4	1		3	8	1
8	2	gemma4			3	9	1
9	2	ministral-3:14b	1		1	10	1
10	2	gemini-3.1-flash-lite	6	1	4		2
11	3	ministral-3:14b	1			12	
12	3	gemma4			8	3	2
13	3	ministral-3:14b	1		5	6	1
14	3	gemini-3.1-flash-lite	6	3	1		3
15	3	gemma4	2	1	4	1	5
16	4	gemini-3.1-flash-lite			3	5	5
17	4	gemma4	1		1	8	3
18	4	ministral-3:14b	1		3	9	
19	4	gemma4			7	5	1
20	4	ministral-3:14b		1		10	2
21	5	gemma4	1	1		9	2
22	5	ministral-3:14b			1	10	2
23	5	gemma4		1	5	6	1
24	5	ministral-3:14b	3	1	5	2	2
25	5	gemini-3.1-flash-lite	4	1	5		3

A seleção das imagens submetidas à avaliação humana adotou o critério de Amostragem Representativa baseada na Mediana de Erro. Para cada mapa candidato, calculou-se um Índice de Desvio S, definido pela soma dos desvios absolutos

entre as entidades esperadas e geradas, acrescida da quantidade de entidades inalcançáveis identificadas pelo validador:

$$S = |I_{esp} - I_{ger}| + |B_{esp} - B_{ger}| + U$$

em que I_{esp} e I_{ger} representam, respectivamente, o número de inimigos esperados e gerados; B_{esp} e B_{ger} representam o número de baús esperados e gerados; e U representa a quantidade de entidades inacessíveis pelo algoritmo de navegação. Em vez de selecionar manualmente os mapas visualmente mais atrativos, os candidatos foram ordenados por esse índice dentro de seus respectivos agrupamentos, e foram escolhidos mapas situados na região mediana da distribuição de erro. Em casos de empate, aplicou-se seleção aleatória simples entre os candidatos empatados, mitigando viés de escolha subjetiva pelo pesquisador.

A Figura 14 evidencia uma divergência relevante entre a validação automatizada e a percepção humana: os modelos de execução local (Gemma4 e Ministral-3:14b), apesar de terem cometido mais infrações sintáticas, obtiveram uma avaliação estética global de 3,54, consideravelmente superior à nota 2,83 do Gemini.

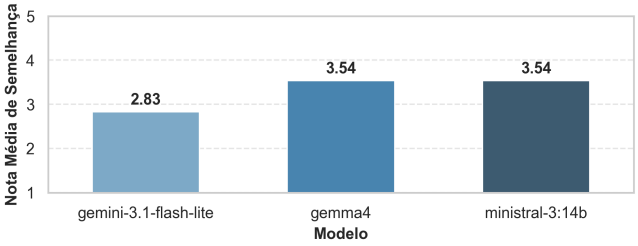

Figura 14. Avaliação Média Global

O Espectro de Respostas (Figura 15) corrobora este dado, indicando que os modelos abertos conseguiram gerar entre 40,0% e 53,8% de mapas considerados com "Boa semelhança" em relação aos temas fornecidos, ao passo que o modelo em nuvem frequentemente gerava layouts neutros ou semanticamente desconexos (representando 35,4% dos votos de "Nenhuma semelhança").

Por fim, a Figura 16 demonstra que os modelos mantiveram consistência de interpretação temática ao longo de todos os níveis, provando que a flexibilidade criativa das configurações probabilísticas da IA compensou as restrições arquiteturais da GPCE, resultando em masmorras perceptivelmente mais orgânicas para o usuário humano.

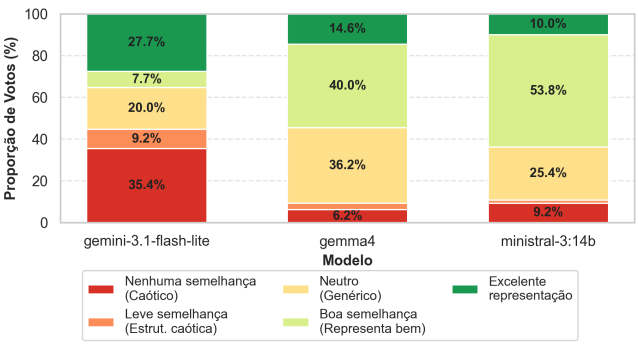

Figura 15. Espectro de Respostas da Avaliação Humana

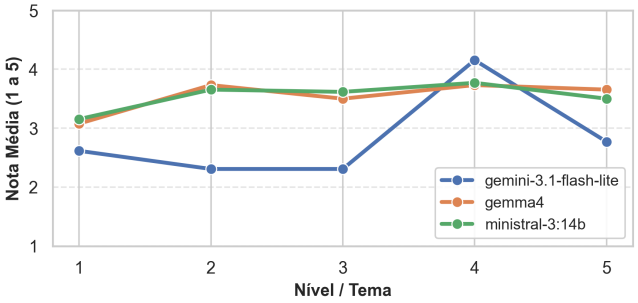

Figura 16. Evolução da Compreensão Temática por Nível

8 Conclusão

Este trabalho propôs e validou uma arquitetura híbrida para a Geração Procedural de Conteúdo Espacial, unindo a capacidade criativa e interpretativa das LLMs com o rigor estrutural de um Agente Validador Autônomo baseado no algoritmo de busca A^*. A pesquisa buscou responder se era viável utilizar IAs textuais para produzir mapas que fossem simultaneamente jogáveis e tematicamente coerentes para o gênero *dungeon crawler*.

A resposta empírica obtida é afirmativa, porém estritamente condicionada à presença da camada de triagem algorítmica. Os testes demonstraram que a utilização da saída bruta das LLMs para instanciar níveis é inviável. Os modelos de linguagem, ao processarem dados como sequências unidimensionais de *tokens*, sofrem do fenômeno da "cegueira espacial" e de severas deficiências na contagem de instâncias matemáticas. Isso resulta na frequente alucinação de entidades presas em enclaves murados e na ausência de caminhos funcionais para a conclusão da fase.

A eficácia do sistema reside fundamentalmente no ciclo de resiliência. O Agente Validador provou ser indispensável, operando silenciosamente para interceptar e descartar *layouts* injogáveis, solicitando novas iterações até entregar uma matriz perfeitamente conectada e funcional para a renderização do jogo. Adicionalmente, a análise qualitativa revelou que, apesar das infrações sintáticas, os modelos de execução local superaram o modelo em nuvem na compreensão e na fidelidade atmosférica dos temas sugeridos, entregando masmorras com designs mais orgânicos.

Contudo, uma observação crítica emerge da análise de desempenho e latência. Com tempos de inferência oscilando entre 40 e 50 segundos em hardware de consumo padrão, além da necessidade de múltiplas retentativas para contornar falhas estruturais, a abordagem se apresenta, no momento, inviável para a geração dinâmica de fases em tempo real durante a sessão de jogo do usuário. Em contrapartida, o método se consagra como uma ferramenta extremamente eficaz e promissora para a fase de pré-produção, atuando como um assistente autônomo (*co-pilot*) de *Level Design*. Desenvolvedores podem utilizar a arquitetura para gerar e validar centenas de matrizes baseadas em prompts criativos offline, acelerando drasticamente o fluxo de trabalho de criação de mundos, retendo e exportando apenas os melhores resultados para o produto final.

Também se observa que os resultados obtidos estão condicionados ao porte e à disponibilidade dos modelos avaliados. É plausível supor que LLMs mais robustas, com maior nú-

mero de parâmetros, melhor alinhamento por instruções e maior capacidade de contexto, apresentem melhor aderência ao formato JSON, às restrições de contagem e às relações espaciais implícitas no prompt. Testes exploratórios informais realizados com modelos comerciais mais avançados, por meio de interfaces conversacionais, sugeriram melhora perceptível na qualidade dos mapas gerados. Entretanto, esses testes não foram incorporados à análise experimental por não seguirem o mesmo protocolo controlado de execução, coleta em CSV, repetição sistemática e acesso via API utilizado nos modelos avaliados. Portanto, a comparação com modelos maiores permanece como hipótese promissora, mas não como evidência quantitativa deste trabalho.

Para trabalhos futuros, sugere-se a exploração de Modelos Multimodais que consigam interpretar a geometria bidimensional como imagem, em vez de texto plano; a substituição do validador baseado no $A*$ por agentes testadores treinados com Aprendizado por Reforço Profundo (*Deep Reinforcement Learning*); e a otimização de *prompts* dinâmicos capazes de injetar o histórico de falhas da tentativa anterior na nova requisição, ensinando a IA a não repetir o mesmo bloqueio espacial.

Declaração de Uso de Ferramentas de Inteligência Artificial Generativa

Ferramentas de inteligência artificial generativa foram utilizadas como apoio à escrita acadêmica deste trabalho, auxiliando na reorganização de parágrafos, refinamento da redação, sugestões de estrutura textual e revisão gramatical. Essas ferramentas não foram empregadas para a concepção da pesquisa, definição da metodologia, implementação do protótipo, execução dos experimentos, análise dos resultados ou formulação das conclusões. Todo o conteúdo foi revisado, validado e aprovado pelo autor, que assume integral responsabilidade pela precisão, integridade e originalidade do trabalho.

Referências

Barbosa, J. V. D. (2024). Ia em jogos: aplicações, métodos e como tornar melhor a experiência do jogador. Monografia, Faculdade de Tecnologia, Universidade Estadual de Campinas, Campinas, SP, Brasil.

Bento, D. V. (2018). Usando ia para realizar ajustes de dificuldade dinâmicos em um sistema de batalha em turnos. In *Anais do XVII Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames) - Trilha de Arte & Design*, Porto Alegre, RS, Brasil.

Canedo, G. N. (2022). Criação de mapas utilizando geração procedural. Trabalho de conclusão de curso, Escola Politécnica, Pontifícia Universidade Católica de Goiás, Goiânia, GO, Brasil.

Compton, K. and Mateas, M. (2021). Procedural level design for platform games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 109–111. AAAI Press. DOI: https://doi.org/10.1609/aiide.v2i1.18755.

de Oliveira, K. O. (2022). Utilização de regressão para a predição de mapas gerados proceduralmente. Master's thesis, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, RS, Brasil.

de Oliveira Barbosa Leite, G. and de Lima, E. S. (2015). Geração procedural de mapas para jogos 2d. In *Proceedings of the XIV Brazilian Symposium on Games and Digital Entertainment (SBGames)*, Teresina, PI, Brazil.

de Pontes, R. G. and Gomes, H. M. (2020). Evolutionary procedural content generation for an endless platform game. In *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 80–89, Recife, PE, Brazil. DOI: 10.1109/SBGames51465.2020.00021.

Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 9(1). DOI: 10.1145/2422956.2422957.

Jang, J., Ye, S., and Seo, M. (2023). Can large language models truly understand prompts? a case study with negated prompts. In Albalak, A., Zhou, C., Raffel, C., Ramachandran, D., Ruder, S., and Ma, X., editors, *Proceedings of The 1st Transfer Learning for Natural Language Processing Workshop*, volume 203 of *Proceedings of Machine Learning Research*, pages 52–62. PMLR.

Kim, J. W., Han, D. H., Park, D. B., Min, K. J., Na, C., Won, S. K., and Park, G. N. (2010). The relationships between online game player biogenetic traits, playing time, and the genre of the game being played. *Psychiatry Investigation*, 7(1):17–23. DOI: 10.4306/pi.2010.7.1.17.

Kühr, K. V. (2018). O uso de ia para criação procedural de conteúdo espacial em jogos. Trabalho de conclusão de curso, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil.

Maleki, M. F. and Zhao, R. (2024). Procedural content generation in games: A survey with insights on emerging LLM integration. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 20, pages 167–178. AAAI Press.

Medeiros, P. (2023). Desenvolvimento de um criador de mapas multi-plataforma para jogos digitais. B.S. thesis, Universidade Federal do Rio Grande do Norte.

Medeiros, T. D. C. d. and Rodrigues, G. M. d. C. (2025). Análise de métodos de geração procedural de níveis em jogos roguelike 2d. Trabalho de conclusão de curso (bacharelado), Universidade Federal do Rio de Janeiro. Orientador: Geraldo Bonorino Xexéo.

Rolim, F. M. (2023). Llms e ia generativa em jogos. Trabalho de conclusão de curso, Instituto de Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

Salton, G., Ross, R., and Kelleher, J. (2017). Attentive language models. In Kondrak, G. and Watanabe, T., editors, *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 441–450, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Souza, R., Junior, B., Foss, L., Cavalheiro, G., and Cavalheiro, S. (2019). Geração procedural de mapas dungeon crawl baseada em gramática de grafos para uso em jogos roguelike. In *Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 193–203, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wscad.2019.8668.

Truong, T. H., Baldwin, T., Verspoor, K., and Cohn, T. (2023). Language models are not naysayers: an analysis of language models on negation benchmarks. In Palmer, A. and Camacho-collados, J., editors, *Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023)*, pages 101–114, Toronto, Canada. Association for Computational Linguistics. DOI: 10.18653/v1/2023.starsem-1.10.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Viana, B. M. d. F. (2019). Geração automática de níveis de masmorras com barreiras para jogos digitais. B.S. thesis, Universidade Federal do Rio Grande do Norte.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.-Y., and Wen, J.-R. (2025). A survey of large language models.

Zhou, Z., Zhou, T., Jia, R., and May, J. (2026). How language models process negation.